CHARAGRAM: Embedding Words and Sentences via Character n-grams

John Wieting   Mohit Bansal   Kevin Gimpel   Karen Livescu
Toyota Technological Institute at Chicago, Chicago, IL, 60637, USA
{jwieting,mbansal,kgimpel,klivescu}@ttic.edu

Abstract

We present CHARAGRAM embeddings, a simple approach for learning character-based compositional models to embed textual sequences. A word or sentence is represented using a character n-gram count vector, followed by a single nonlinear transformation to yield a low-dimensional embedding. We use three tasks for evaluation: word similarity, sentence similarity, and part-of-speech tagging. We demonstrate that CHARAGRAM embeddings outperform more complex architectures based on character-level recurrent and convolutional neural networks, achieving new state-of-the-art performance on several similarity tasks.[1]

1   Introduction

Representing textual sequences such as words and sentences is a fundamental component of natural language understanding systems. Many functional architectures have been proposed to model compositionality in word sequences, ranging from simple averaging (Mitchell and Lapata, 2010; Iyyer et al., 2015) to functions with rich recursive structure (Socher et al., 2011; Zhu et al., 2015; Tai et al., 2015; Bowman et al., 2016). Most work uses words as the smallest units in the compositional architecture, often using pretrained word embeddings or learning them specifically for the task of interest (Tai et al., 2015; He et al., 2015).

Some prior work has found benefit from using character-based compositional models that encode arbitrary character sequences into vectors. Examples include recurrent neural networks (RNNs) and convolutional neural networks (CNNs) on character sequences, showing improvements for several NLP tasks (Ling et al., 2015a; Kim et al., 2015; Ballesteros et al., 2015; dos Santos and Guimarães, 2015). By sharing subword information across words, character models have the potential to better represent rare words and morphological variants.

Our approach, CHARAGRAM, uses a much simpler functional architecture. We represent a character sequence by a vector containing counts of character n-grams, inspired by Huang et al. (2013). This vector is embedded into a low-dimensional space using a single nonlinear transformation. This can be interpreted as learning embeddings of character n-grams, which are learned so as to produce effective sequence embeddings when a summation is performed over the character n-grams in the sequence.

We consider three evaluations: word similarity, sentence similarity, and part-of-speech tagging. On multiple word similarity datasets, CHARAGRAM outperforms RNNs and CNNs, achieving state-of-the-art performance on SimLex-999 (Hill et al., 2015). When evaluated on a large suite of sentence-level semantic textual similarity tasks, CHARAGRAM embeddings again outperform the RNN and CNN architectures as well as the PARAGRAM-PHRASE embeddings of Wieting et al. (2016). We also consider English part-of-speech (POS) tagging using the bidirectional long short-term memory tagger of Ling et al. (2015a). The three architectures reach similar performance, though CHARAGRAM converges fastest to high accuracy.

---

[1]Trained models and code are available at http://ttic.uchicago.edu/~wieting.

We perform extensive analysis of our CHARA-GRAM embeddings. We find large gains in performance on rare words, showing the empirical benefit of subword modeling. We also compare performance across different character n-gram vocabulary sizes, finding that the semantic tasks benefit far more from large vocabularies than the syntactic task. However, even for challenging semantic similarity tasks, we still see strong performance with only a few thousand character n-grams.

Nearest neighbors show that CHARAGRAM embeddings simultaneously address differences due to spelling variation, morphology, and word choice. Inspection of embeddings of particular character n-grams reveals etymological links; e.g., *die* is close to *mort*. We release our resources to the community in the hope that CHARAGRAM can provide a strong baseline for subword-aware text representation.

2 Related Work

We first review work on using subword information in word embedding models. The simplest approaches append subword features to word embeddings, letting the model learn how to use the subword information for particular tasks. Some added knowledge-based morphological features to word representations (Alexandrescu and Kirchhoff, 2006; El-Desoky Mousa et al., 2013). Others learned embeddings jointly for subword units and words, defining simple compositional architectures (often based on addition) to create word embeddings from subword embeddings (Lazaridou et al., 2013; Botha and Blunsom, 2014; Qiu et al., 2014; Chen et al., 2015).

A recent trend is to use richer functional architectures to convert character sequences into word embeddings. Luong et al. (2013) used recursive models to compose morphs into word embeddings, using unsupervised morphological analysis. Ling et al. (2015a) used a bidirectional long short-term memory (LSTM) RNN on characters to embed arbitrary word types, showing strong performance for language modeling and POS tagging. Ballesteros et al. (2015) used this model to represent words for dependency parsing. Several have used character-level RNN architectures for machine translation, whether for representing source or target words (Ling et al., 2015b; Luong and Man-

ning, 2016), or for generating entire translations character-by-character (Chung et al., 2016).

Sutskever et al. (2011) and Graves (2013) used character-level RNNs for language modeling. Others trained character-level RNN language models to provide features for NLP tasks, including tokenization and segmentation (Chrupała, 2013; Evang et al., 2013), and text normalization (Chrupała, 2014).

CNNs with character n-gram filters have been used to embed arbitrary word types for several tasks, including language modeling (Kim et al., 2015), part-of-speech tagging (dos Santos and Zadrozny, 2014), named entity recognition (dos Santos and Guimarães, 2015), text classification (Zhang et al., 2015), and machine translation (Costa-Jussà and Fonollosa, 2016). Combinations of CNNs and RNNs on characters have also been explored (Józefowicz et al., 2016).

Most closely-related to our approach is the DSSM (instantiated variously as "deep semantic similarity model" or "deep structured semantic model") developed by Huang et al. (2013). For an information retrieval task, they represented words using feature vectors containing counts of character n-grams. Sperr et al. (2013) used a very similar technique to represent words in neural language models for machine translation. Our CHARAGRAM embeddings are based on this same idea. We show this strategy to be extremely effective when applied to both words and sentences, outperforming character LSTMs like those used by Ling et al. (2015a) and character CNNs like those from Kim et al. (2015).

3 Models

We now describe models that embed textual sequences using their characters, including our CHARAGRAM model and the baselines that we compare to. We denote a character-based textual sequence by $x = \langle x_1, x_2, ..., x_m \rangle$, which includes space characters between words as well as special start-of-sequence and end-of-sequence characters. We use x_i^j to denote the subsequence of characters from position i to position j inclusive, i.e., $x_i^j = \langle x_i, x_{i+1}, ..., x_j \rangle$, and we define $x_i^i = x_i$.

Our CHARAGRAM model embeds a character sequence x by adding the vectors of its character n-

grams followed by an elementwise nonlinearity:

$$g_{\text{CHAR}}(x) = h\left(\boldsymbol{b} + \sum_{i=1}^{m+1} \sum_{j=1+i-k}^{i} \mathbb{I}\left[x_j^i \in V\right] W^{x_j^i}\right)$$
(1)

where h is a nonlinear function, $\boldsymbol{b} \in \mathbb{R}^d$ is a bias vector, k is the maximum length of any character n-gram, $\mathbb{I}[p]$ is an indicator function that returns 1 if p is true and 0 otherwise, V is the set of character n-grams included in the model, and $W^{x_j^i} \in \mathbb{R}^d$ is the vector for character n-gram x_j^i.

The set V is used to restrict the model to a predetermined set (vocabulary) of character n-grams. Below, we compare several choices for V. The number of parameters in the model is $d + d|V|$. This model is based on the letter n-gram hashing technique developed by Huang et al. (2013). One can also view Eq. (1) (as they did) as first populating a vector of length $|V|$ with counts of character n-grams followed by a nonlinear transformation.

We compare the CHARAGRAM model to two other models. First we consider LSTM architectures (Hochreiter and Schmidhuber, 1997) over the character sequence x, using the version from Gers et al. (2003). We use a forward LSTM over the characters in x, then take the final LSTM hidden vector as the representation of x. Below we refer to this model as "charLSTM."

We also compare to convolutional neural network (CNN) architectures, which we refer to below as "charCNN." We use the architecture from Kim (2014) with a single convolutional layer followed by an optional fully-connected layer. We use filters of varying lengths of character n-grams, using two primary configurations of filter sets, one of which is identical to that used by Kim et al. (2015). Each filter operates over the entire sequence of character n-grams in x and we use max pooling for each filter. We tune over the choice of nonlinearity for both the convolutional filters and for the optional fully-connected layer. We give more details below about filter sets, n-gram lengths, and nonlinearities.

We note that using character n-gram convolutional filters is similar to our use of character n-grams in the CHARAGRAM model. The difference is that, in the CHARAGRAM model, the n-gram must match exactly for its vector to affect the representa-

tion, while in the CNN each filter will affect the representation of all sequences (depending on the nonlinearity being used). So the CHARAGRAM model is able to learn precise vectors for particular character n-grams with specific meanings, while there is pressure for the CNN filters to capture multiple similar patterns that recur in the data. Our qualitative analysis shows the specificity of the learned character n-gram vectors learned by the CHARAGRAM model.

4 Experiments

We perform three sets of experiments. The goal of the first two (Section 4.1) is to produce embeddings for textual sequences such that the embeddings for paraphrases have high cosine similarity. Our third evaluation (Section 4.2) is a classification task, and follows the setup of the English part-of-speech tagging experiment from Ling et al. (2015a).

4.1 Word and Sentence Similarity

We compare the ability of our models to capture semantic similarity for both words and sentences. We train on noisy paraphrase pairs from the Paraphrase Database (PPDB; Ganitkevitch et al., 2013) with an L_2 regularized contrastive loss objective function, following the training procedure of Wieting et al. (2015) and Wieting et al. (2016). More details are provided in the supplementary material.

4.1.1 Datasets

For word similarity, we focus on two of the most commonly used datasets for evaluating semantic similarity of word embeddings: WordSim-353 (WS353) (Finkelstein et al., 2001) and SimLex-999 (SL999) (Hill et al., 2015). We also evaluate our best model on the Stanford Rare Word Similarity Dataset (Luong et al., 2013).

For sentence similarity, we evaluate on a diverse set of 22 textual similarity datasets, including all datasets from every SemEval semantic textual similarity (STS) task from 2012 to 2015. We also evaluate on the SemEval 2015 Twitter task (Xu et al., 2015) and the SemEval 2014 SICK Semantic Relatedness task (Marelli et al., 2014). Given two sentences, the aim of the STS tasks is to predict their similarity on a 0-5 scale, where 0 indicates the sentences are on different topics and 5 indicates that they are completely equivalent.

Each STS task consists of 4-6 datasets covering a wide variety of domains, including newswire, tweets, glosses, machine translation outputs, web forums, news headlines, image and video captions, among others. Most submissions for these tasks use supervised models that are trained and tuned on provided training data or similar datasets from older tasks. Further details are provided in the official task descriptions (Agirre et al., 2012; Agirre et al., 2013; Agirre et al., 2014; Agirre et al., 2015).

4.1.2 Preliminaries

For training data, we use pairs from PPDB. For word similarity experiments, we train on word pairs and for sentence similarity, we train on phrase pairs. PPDB comes in different sizes (S, M, L, XL, XXL, and XXXL), where each larger size subsumes all smaller ones. The pairs in PPDB are sorted by a confidence measure and so the smaller sets contain higher precision paraphrases. PPDB is derived automatically from naturally-occurring bilingual text, and versions of PPDB have been released for many languages without the need for any manual annotation (Ganitkevitch and Callison-Burch, 2014).

Before training the CHARAGRAM model, we need to populate V, the vocabulary of character n-grams included in the model. We obtain these from the training data used for the final models in each setting, which is either the lexical or phrasal section of PPDB XXL. We tune over whether to include the full sets of character n-grams in these datasets or only those that appear more than once.

When extracting n-grams, we include spaces and add an extra space before and after each word or phrase in the training and evaluation data to ensure that the beginning and end of each word is represented. We note that strong performance can be obtained using far fewer character n-grams; we explore the effects of varying the number of n-grams and the n-gram orders in Section 4.4.

We used Adam (Kingma and Ba, 2014) with a learning rate of 0.001 to learn the parameters in the following experiments.

4.1.3 Word Embedding Experiments

Training and Tuning For hyperparameter tuning, we used one epoch on the lexical section of PPDB XXL, which consists of 770,007 word pairs. We

Model	Tuned on	WS353	SL999
charCNN	SL999	26.31	30.64
	WS353	33.19	16.73
charLSTM	SL999	48.27	54.54
	WS353	51.43	48.83
CHARAGRAM	SL999	53.87	**63.33**
	WS353	**58.35**	60.00
inter-annotator agreement	-	75.6	78

Table 1: Word similarity results (Spearman's $\rho \times 100$). The inter-annotator agreement is the average Spearman's ρ between a single annotator and the average of all others.

used either WS353 or SL999 for model selection (reported below). We then took the selected hyperparameters and trained for 50 epochs to ensure that all models had a chance to converge.

Full details of our tuning procedure are provided in the supplementary material. In short, we tuned all models thoroughly, tuning the activation functions for CHARAGRAM and charCNN, as well as the regularization strength, mini-batch size, and sampling type for all models. For charCNN, we experimented with two filter sets: one uses 175 filters for each n-gram size $\in \{2, 3, 4\}$, and the other uses the set of filters from Kim et al. (2015), consisting of 25 filters of size 1, 50 of size 2, 75 of size 3, 100 of size 4, 125 of size 5, and 150 of size 6. We also experimented with using dropout (Srivastava et al., 2014) on the inputs to the final layer of charCNN in place of L_2 regularization, as well as removing the last feedforward layer. Neither variation significantly improved performance on our suite of tasks for word or sentence similarity. However, using more filters does improve performance, apparently linearly with the square of the number of filters.

Architecture Comparison The results are shown in Table 1. The CHARAGRAM model outperforms both the charLSTM and charCNN models, and also outperforms recent strong results on SL999.

We also found that the charCNN and charLSTM models take far more epochs to converge than the CHARAGRAM model. We noted this trend across experiments and explore it further in Section 4.3.

Comparison to Prior Work We found that performance of CHARAGRAM on word similarity tasks can be improved by using more character n-grams. This is explored in Section 4.4. Our best result from these experiments was obtained with the largest

Model	SL999
Hill et al. (2014)	52
Schwartz et al. (2015)	56
Faruqui and Dyer (2015)	58
Wieting et al. (2015)	66.7
CHARAGRAM (large)	**70.6**

Table 2: Spearman's $\rho \times 100$ on SL999. CHARAGRAM (large) refers to the CHARAGRAM model described in Section 4.4. This model contains 173,881 character n-grams, more than the 100,283 in the CHARAGRAM model used in Table 1.

model we considered, which contains 173,881 n-gram embeddings. When using WS353 for model selection and training for 25 epochs, this model achieves 70.6 on SL999. To our knowledge, this is the best result reported on SL999 in this setting; Table 2 shows comparable recent results. Note that a higher SL999 number is reported by Mrkšić et al. (2016), but the setting is not comparable to ours as they started with embeddings tuned on SL999.

Lastly, we evaluated our model on the Stanford Rare Word Similarity Dataset (Luong et al., 2013), using SL999 for model selection. We obtained a Spearman's ρ of 47.1, which outperforms the 41.8 result from Soricut and Och (2015) and is competitive with the 47.8 reported by Pennington et al. (2014), which used a 42B-token corpus for training.

4.1.4 Sentence Embedding Experiments

Training and Tuning We did initial training of our models using one pass through PPDB XL, which consists of 3,033,753 unique phrase pairs. Following Wieting et al. (2016), we use the annotated phrase pairs developed by Pavlick et al. (2015) as our validation set, using Spearman's ρ to rank the models. We then take the highest performing models and train on the 9,123,575 unique phrase pairs in the phrasal section of PPDB XXL for 10 epochs.

For all experiments, we fix the mini-batch size to 100, the margin δ to 0.4, and use MAX sampling (see supplementary material). For CHARAGRAM, V contains all 122,610 character n-grams ($n \in \{2, 3, 4\}$) in the PPDB XXL phrasal section. Other tuning settings are the same as Section 4.1.3.

For another baseline, we train the PARAGRAM-PHRASE model of Wieting et al. (2016), tuning its regularization strength over $\{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$. The PARAGRAM-PHRASE model simply uses word averaging as its

composition function, but outperforms many more complex models.

In this section, we refer to our model as CHARAGRAM-PHRASE because the input is a character sequence containing multiple words rather than only a single word as in Section 4.1.3. Since the vocabulary V is defined by the training data sequences, the CHARAGRAM-PHRASE model includes character n-grams that span multiple words, permitting it to capture some aspects of word order and word co-occurrence, which the PARAGRAM-PHRASE model is unable to do.

We encountered difficulties training the charLSTM and charCNN models for this task. We tried several strategies to improve their chance at convergence, including clipping gradients, increasing training data, and experimenting with different optimizers and learning rates. We found success by using the original (confidence-based) ordering of the PPDB phrase pairs for the initial epoch of learning, then shuffling them for subsequent epochs. This is similar to curriculum learning (Bengio et al., 2009). The higher-confidence phrase pairs tend to be shorter and have many overlapping words, possibly making them easier to learn from.

Results An abbreviated version of the sentence similarity results is shown in Table 3; the supplementary material contains the full results. For comparison, we report performance for the median (50%), third quartile (75%), and top-performing (Max) systems from the shared tasks. We observe strong performance for the CHARAGRAM-PHRASE model. It always does better than the charCNN and charLSTM models, and outperforms the PARAGRAM-PHRASE model on 15 of the 22 tasks. Furthermore, CHARAGRAM-PHRASE matches or exceeds the top-performing task-tuned systems on 5 tasks, and is within 0.003 on 2 more. The charLSTM and charCNN models are significantly worse, with the charCNN being the better of the two and beating PARAGRAM-PHRASE on 4 of the tasks.

We emphasize that there are many other models that could be compared to, such as an LSTM over word embeddings. This and many other models were explored by Wieting et al. (2016). Their PARAGRAM-PHRASE model, which simply learns word embeddings within an averaging composition

Dataset	50%	75%	Max	charCNN	charLSTM	PARAGRAM-PHRASE	CHARAGRAM-PHRASE
STS 2012 Average	54.5	59.5	70.3	56.5	40.1	58.5	**66.1**
STS 2013 Average	45.3	51.4	65.3	47.7	30.7	**57.7**	57.2
STS 2014 Average	64.7	71.4	76.7	64.7	46.8	71.5	**74.7**
STS 2015 Average	70.2	75.8	80.2	66.0	45.5	75.7	**76.1**
2014 SICK	71.4	79.9	82.8	62.9	50.3	**72.0**	70.0
2015 Twitter	49.9	52.5	61.9	48.6	39.9	52.7	**53.6**
Average	59.7	65.6	73.6	59.2	41.9	66.2	**68.7**

Table 3: Results on SemEval textual similarity datasets (Pearson's $r \times 100$). The highest score in each row is in boldface (omitting the official task score columns). The last row shows the average performance over all 22 textual similarity datasets

Model	Accuracy (%)
charCNN	97.02
charLSTM	96.90
CHARAGRAM	96.99
CHARAGRAM (2-layer)	**97.10**

Table 4: Results on part-of-speech tagging.

function, was among their best-performing models. We used this model in our experiments as a strongly-performing representative of their results.

Lastly, we note other recent work that considers a similar transfer learning setting. The Fast-Sent model (Hill et al., 2016) uses the 2014 STS task in its evaluation and reports an average Pearson's r of 61.3. On the same data, the C-PHRASE model (Pham et al., 2015) has an average Pearson's r of 65.7.[2] Both results are lower than the 74.7 achieved by CHARAGRAM-PHRASE on this dataset.

4.2 POS Tagging Experiments

We now consider part-of-speech (POS) tagging, since it has been used as a testbed for evaluating architectures for character-level word representations. It also differs from semantic similarity, allowing us to evaluate our architectures on a syntactic task. We replicate the POS tagging experimental setup of Ling et al. (2015a). Their model uses a bidirectional LSTM over character embeddings to represent words. They then use the resulting word representations in another bidirectional LSTM that predicts the tag for each word. We replace their character bidirectional LSTM with our three architectures: charCNN, charLSTM, and CHARAGRAM.

We use the Wall Street Journal portion of the Penn Treebank, using Sections 1-18 for training, 19-21 for tuning, and 22-24 for testing. We set the dimensionality of the character embeddings to 50 and that of

the (induced) word representations to 150. For optimization, we use stochastic gradient descent with a mini-batch size of 100 sentences. The learning rate and momentum are set to 0.2 and 0.95 respectively. We train the models for 50 epochs, again to ensure that all models have an opportunity to converge.

The other settings for our models are mostly the same as for the word and sentence experiments (Section 4.1). We again use character n-grams with $n \in \{2, 3, 4\}$, tuning over whether to include all 54,893 in the training data or only those that occur more than once. However, there are two minor differences from the previous sections. First, we add a single binary feature to indicate if the token contains a capital letter. Second, our tuning considers rectified linear units as the activation function for the CHARAGRAM and charCNN architectures.[3]
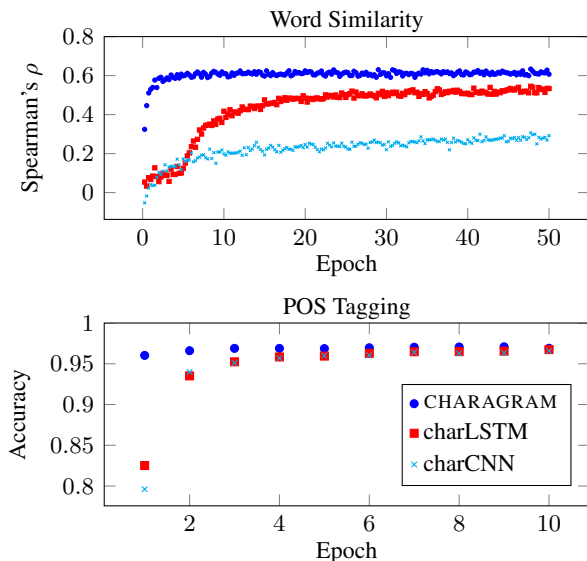
The results are shown in Table 4. Performance is similar across models. We found that adding a second fully-connected 150 dimensional layer to the CHARAGRAM model improved results slightly.[4]

4.3 Convergence

One observation we made during our experiments was that different models converged at significantly different rates. Figure 1 plots the performance of the word similarity and tagging tasks as a function of training epoch. For word similarity, we plot the oracle Spearman's ρ on SL999, while for tagging we plot accuracy on the tuning set. We evaluate every quarter epoch (approximately every 194,252 word pairs) for word similarity and every epoch for tag-

---

[2]Both the results for FastSent and C-PHRASE were computed from Table 4 in (Hill et al., 2016).

[3]We did not consider ReLU for the similarity experiments because the final embeddings are used directly to compute cosine similarities, which led to poor performance when restricting the embeddings to be non-negative.

[4]We also tried adding a second (300 dimensional) layer for the word and sentence similarity models and found it to hurt performance.

Figure 1: Plots of performance versus training epoch for word similarity and POS tagging.

Task	# n-grams	2	2,3	2,3,4	2,3,4,5	2,3,4,5,6
POS Tagging	100	95.52	96.09	96.15	96.13	96.16
	1,000	96.72	96.86	96.97	97.02	97.03
	50,000	96.81	97.00	97.02	97.04	**97.09**
Word Similarity	100	6.2	7.0	7.7	9.1	8.8
	1,000	15.2	33.0	38.7	43.2	43.9
	50,000	14.4	52.4	67.8	69.2	**69.5**
Sentence Similarity	100	40.2	33.8	32.5	31.2	29.8
	1,000	50.1	60.3	58.6	56.6	55.6
	50,000	45.7	64.7	**66.6**	63.0	61.3

Table 5: Results of using different numbers and different combinations of character n-grams.

and 763,200 parameters respectively (including 134 character embeddings for each).

However, for a given sequence, very few parameters in the CHARAGRAM model are actually used. For charCNN and charLSTM, by contrast, all parameters are used except character embeddings for characters not present in the sequence. For a 100-character sequence, the 300-dimensional CHARAGRAM model uses approximately 90,000 parameters, about one-tenth of those used by charCNN and charLSTM for the same sequence.

We performed a series of experiments to investigate how the CHARAGRAM and CHARAGRAM-PHRASE models perform with different numbers and lengths of character n-grams. For a given k, we took the k most frequent character n-grams for each value of n in use. We experimented with k values in $\{100, 1000, 50000\}$. If there were fewer than k unique character n-grams for a given n, we used all of them. For these experiments, we did very little tuning, setting the regularization strength to 0 and only tuning the activation function. For word similarity, we report performance on SL999 after 5 training epochs on the lexical section of PPDB XXL. For sentence similarity, we report the average Pearson's r over all 22 datasets after 5 training epochs on the phrasal section of PPDB XL. For tagging, we report accuracy on the tuning set after 50 training epochs.

The results are shown in Table 5. When using extremely small models with only 100 n-grams of each order, we still see relatively strong performance on tagging. However, the similarity tasks require far more n-grams to yield strong performance. Using 1000 n-grams clearly outperforms 100, and 50,000 n-grams performs best. We also found that models converged more quickly on tagging than on the similarity tasks. We suspect this is due to differences in task complexity. In tagging, the model does not

ging. We only show the first 10 epochs of training in the tagging plot.

The plots show that the CHARAGRAM model converges quickly to high performance. The charCNN and charLSTM models take many more epochs to converge. Even with tagging, which uses a very high learning rate, CHARAGRAM converges significantly faster than the others. For word similarity, it appears that charCNN and charLSTM are still slowly improving at the end of 50 epochs. This suggests the possibility that these models could eventually surpass CHARAGRAM with more epochs. However, due to the large training sets available from PPDB and the computational requirements of these architectures, we were unable to explore the regime of training for many epochs. We conjecture that slow convergence could also be the reason for the inferior performance of LSTMs for similarity tasks as reported by Wieting et al. (2016).

4.4 Model Size Experiments

The default setting for our CHARAGRAM and CHARAGRAM-PHRASE models is to use all character bigram, trigrams, and 4-grams that occur in the training data at least C times, tuning C over the set $\{1, 2\}$. This results in a large number of parameters, which could be seen as an unfair advantage over the comparatively smaller charCNN and charLSTM similarity models, which have up to 881,025

need to learn all facets of each word's semantics; it only needs to map a word to its syntactic categories. Therefore, simple surface-level features like affixes can help tremendously. However, learning representations that reflect detailed differences in word meaning is a more fine-grained endeavor and this is presumably why larger models are needed and convergence is slower.

5 Analysis

5.1 Quantitative Analysis

One of our primary motivations for character-based models is to address the issue of out-of-vocabulary (OOV) words, which were found to be one of the main sources of error for the PARAGRAM-PHRASE model from Wieting et al. (2016). They reported a negative correlation (Pearson's r of -0.45) between OOV rate and performance. We took the 12,108 sentence pairs in all 20 SemEval STS tasks and binned them by the total number of unknown words in the pairs.[5] We computed Pearson's r over each bin. The results are shown in Table 6.

Number of Unknown Words	N	PARAGRAM-PHRASE	CHARAGRAM-PHRASE
0	11,292	71.4	**73.8**
1	534	68.8	**78.8**
2	194	66.4	**72.8**
≥ 1	816	68.6	**77.9**
≥ 0	12,108	71.0	**74.0**

Table 6: Performance (Pearson's $r \times 100$) as a function of the number of unknown words in the sentence pairs over all 20 SemEval STS datasets. N is the number of sentence pairs.

The CHARAGRAM-PHRASE model has better performance for each number of unknown words. The PARAGRAM-PHRASE model degrades when more unknown words are present, presumably because it is forced to use the same unknown word embedding for all unknown words. The CHARAGRAM-PHRASE model has no notion of unknown words, as it can embed any character sequence.

We next investigated the sensitivity of the two models to length, as measured by the maximum

of the lengths of the two sentences in a pair. We binned all of the 12,108 sentence pairs in the 20 SemEval STS tasks by length and then again found the Pearson's r for both the PARAGRAM-PHRASE and CHARAGRAM-PHRASE models. The results are shown in Table 7.

Max Length	N	PARAGRAM-PHRASE	CHARAGRAM-PHRASE
≤ 4	71	67.9	**72.9**
5	216	71.1	**71.9**
6	572	67.0	**69.7**
7	1,097	71.5	**74.0**
8	1,356	74.2	**74.5**
9	1,266	71.7	**72.7**
10	1,010	70.7	**74.2**
11-15	3,143	71.8	**73.7**
16-20	1,559	73.0	**75.1**
≥ 21	1,818	74.5	**75.4**

Table 7: Performance (Pearson's $r \times 100$) as a function of the maximum number of tokens in the sentence pairs over all 20 SemEval STS datasets. N is the number of sentence pairs.

Both models are robust to sentence length, achieving the highest correlations on the longest sentences. We also find that CHARAGRAM-PHRASE outperforms PARAGRAM-PHRASE at all sentence lengths.

5.2 Qualitative Analysis

Bigram	CHARAGRAM-PHRASE	PARAGRAM-PHRASE
not capable	incapable, unable, incapacity	not, capable, stalled
not able	unable, incapable, incapacity	not, able, stalled
not possible	impossible impracticable unable	not, stalled, possible
not sufficient	insufficient, sufficient, inadequate	not, sufficient, stalled
not easy	easy, difficult, tough	not, stalled, easy

Table 8: Nearest neighboring words of selected bigrams under CHARAGRAM-PHRASE and PARAGRAM-PHRASE embeddings.

Aside from OOVs, the PARAGRAM-PHRASE model lacks the ability to model word order or cooccurrence, since it simply averages the words in the sequence. We were interested to see whether CHARAGRAM-PHRASE could handle negation, since it does model limited information about word order (via character n-grams that span multiple words). We made a list of "not" bigrams that could be represented by a single word, then embedded each bigram using both models and did a nearest-neighbor search over a working vocabulary.[6] The results, in Table 8, show how the CHARAGRAM-PHRASE embeddings model negation. In all cases but one, the near-

---

[5]Unknown words were defined as those not present in the 1.7 million unique (case-insensitive) tokens that comprise the vocabulary for the GloVe embeddings available at http://nlp.stanford.edu/projects/glove/. The PARAGRAM-SL999 embeddings, used to initialize the PARAGRAM-PHRASE model, use this same vocabulary.

[6]This has all words in PPDB-XXL, our evaluations, and two other datasets: SST (Socher et al., 2013) and SNLI (Bowman et al., 2015), resulting in 93,217 unique (up-to-casing) tokens.

Word	Nearest Neighbors
vehicals	vehical, vehicles, vehicels, vehicular, cars, vehicle, automobiles, car
serious-looking	serious, grave, acute, serious-minded, seriousness, gravity, serious-faced
near-impossible	impossible, hard/impossible, audacious-impossible, impractical, unable
growths	growth, grow, growing, increases, grows, increase, rise, growls, rising
litered	liter, litering, lited, liters, literate, literature, literary, literal, lite, obliterated
journeying	journey, journeys, voyage, trip, roadtrip, travel, tourney, voyages, road-trip
babyyyyyy	babyyyyyyy, baby, babys, babe, baby.i, babydoll, babycake, darling
adirty	dirty, dirtyyyyyy, filthy, down-and-dirty, dirtying, dirties, ugly, dirty-blonde
refunding	refunds, refunded, refund, repayment, reimbursement, rebate, repay reimbursements, reimburse, repaying, repayments, rebates, rebating, reimburses
professors	professor, professorships, professorship, teachers, professorial, teacher prof., teaches, lecturers, teachings, instructors, headteachers, teacher-student
huge	enormous, tremendous, large, big, vast, overwhelming, immense, giant formidable, considerable, massive, huger, large-scale, great, daunting

Table 9: Nearest neighbors of CHARAGRAM-PHRASE embeddings. Above the double horizontal line are nearest neighbors of words that were not in our training data, and below it are nearest neighbors of words that were in our training data.

est neighbor is a paraphrase for the bigram and the next neighbors are mostly paraphrases as well. The PARAGRAM-PHRASE model, unsurprisingly, is incapable of modeling negation. In all cases, the nearest neighbor is *not*, as it carries much more weight than the word it modifies. The remaining nearest neighbors are either the modified word or *stalled*.

We did two additional nearest neighbor explorations with our CHARAGRAM-PHRASE model. First, we collected nearest neighbors for words that were not in the training data (i.e., PPDB XXL), but were in our working vocabulary. These are shown in the upper part of Table 9. In the second, we collected nearest neighbors of words that were in our training data, shown in the lower part of Table 9.

Several kinds of similarity are being captured simultaneously. One kind is similarity in terms of spelling variation, including misspellings (*vehicals*, *vehicels*) and repetition for emphasis (*babyyyyyy*). Another kind is similarity in terms of morphological variants of a shared root (e.g., *journeying* and *journey*). We also find many synonym relationships without significant amounts of overlapping characters (e.g., *vehicles*, *cars*, *automobiles*). Words in the training data, which tend to be more commonly used, do tend to have higher precision in their nearest neighbors (e.g., neighbors of *huge*). We see occasional mistakes for words that share many characters but are not paraphrases (e.g., *litered*, a likely misspelling of *littered*).

Lastly, since our model learns embeddings for character n-grams, we show an analysis of character n-gram nearest neighbors in Table 10. They appear to be grouped into themes, such as death (row

n-gram	Nearest Neighbors
die	_dy, _die, dead,_dyi, rlif, mort, ecea, rpse, d_aw
foo	_foo,_eat, meal, alim, trit, feed, grai,_din, nutr, toe
pee	peed, hast, spee, fast, mpo_, pace, _vel, loci, ccel
aiv	waiv, aive, boli, epea, ncel, abol, lift, bort, bol
ngu	ngue, uist, ongu, tong, abic, gual, fren, ocab, ingu
_2	2_, _02, _02_,_tw,_dua,_xx,_ii_, xx, o_14, d_2

Table 10: Nearest neighbors of character n-gram embeddings from trained CHARAGRAM-PHRASE model. The underscore indicates a space, which signals the beginning or end of a word.

1), food (row 2), and speed (row 3), but have different granularities. The n-grams in the last row appear in paraphrases of *2*, whereas the second-to-last row shows n-grams in words related to language.

6   Conclusion

We performed a careful empirical comparison of character-based compositional architectures on three NLP tasks. We found a consistent trend: the simplest architecture converges fastest to high performance. These results, coupled with those from Wieting et al. (2016), suggest that practitioners should begin with simple architectures rather than moving immediately to RNNs and CNNs. We release our code and trained models so they can be used by the NLP community for general-purpose, character-based text representation.

References

Eneko Agirre, Mona Diab, Daniel Cer, and Aitor Gonzalez-Agirre. 2012. SemEval-2012 task 6: A pilot on semantic textual similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*. Association for Computational Linguistics.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*.

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2014. SemEval-2014 task 10: Multilingual semantic textual similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*.

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, German Rigau, Larraitz Uria, and Janyce Wiebe. 2015. SemEval-2015 task 2: Semantic textual similarity, English, Spanish and pilot on interpretability. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*.

Andrei Alexandrescu and Katrin Kirchhoff. 2006. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*.

Jan A. Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and D. Christopher Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. In *Proceedings of ACL*.

Xinxiong Chen, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huanbo Luan. 2015. Joint learning of character and word embeddings. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

Grzegorz Chrupała. 2013. Text segmentation with character-level text embeddings. *arXiv preprint arXiv:1309.4628*.

Grzegorz Chrupała. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 2016. A character-level decoder without explicit segmentation for neural machine translation. *arXiv preprint arXiv:1603.06147*.

Marta R. Costa-Jussà and José A. R. Fonollosa. 2016. Character-based neural machine translation. *arXiv preprint arXiv:1603.00810*.

Cicero dos Santos and Victor Guimarães. 2015. Boosting named entity recognition with neural character embeddings. In *Proceedings of the Fifth Named Entity Workshop*.

Cicero dos Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*.

Amr El-Desoky Mousa, Hong-Kwang Jeff Kuo, Lidia Mangu, and Hagen Soltau. 2013. Morpheme-based feature-rich language models using deep neural networks for LVCSR of Egyptian Arabic. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE.

Kilian Evang, Valerio Basile, Grzegorz Chrupała, and Johan Bos. 2013. Elephant: Sequence labeling for word and sentence segmentation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Manaal Faruqui and Chris Dyer. 2015. Non-distributional word vector representations. *arXiv preprint arXiv:1506.05230*.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*. ACM.

Juri Ganitkevitch and Chris Callison-Burch. 2014. The multilingual paraphrase database. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. PPDB: The Paraphrase Database. In *Proceedings of HLT-NAACL*.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. 2003. Learning precise timing with LSTM recurrent networks. *The Journal of Machine Learning Research*, 3.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Hua He, Kevin Gimpel, and Jimmy Lin. 2015. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Felix Hill, Kyunghyun Cho, Sebastien Jean, Coline Devin, and Yoshua Bengio. 2014. Embedding word similarity with neural machine translation. *arXiv preprint arXiv:1412.6448*.

Felix Hill, Roi Reichart, and Anna Korhonen. 2015. SimLex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4).

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning distributed representations of sentences from unlabelled data. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8).

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *CoRR*, abs/1602.02410.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2015. Character-aware neural language models. *CoRR*, abs/1508.06615.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Angeliki Lazaridou, Marco Marelli, Roberto Zamparelli, and Marco Baroni. 2013. Compositional-ly derived representations of morphologically complex words in distributional semantics. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015a. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Wang Ling, Isabel Trancoso, Chris Dyer, and Alan W. Black. 2015b. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*.

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. *arXiv preprint arXiv:1604.00788*.

Thang Luong, Richard Socher, and Christopher Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*.

Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. 2014. SemEval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive Science*, 34(8).

Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. Counter-fitting word vectors to linguistic constraints. *arXiv preprint arXiv:1603.00892*.

Ellie Pavlick, Pushpendre Rastogi, Juri Ganitkevich, Benjamin Van Durme, and Chris Callison-Burch. 2015. PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. *Proceedings of Empirical Methods in Natural Language Processing (EMNLP 2014)*.

Nghia The Pham, Germán Kruszewski, Angeliki Lazaridou, and Marco Baroni. 2015. Jointly optimizing word representations for lexical and sentential tasks with the c-phrase model. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Co-learning of word representations and morpheme representations. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*.

Roy Schwartz, Roi Reichart, and Ari Rappoport. 2015. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Radu Soricut and Franz Och. 2015. Unsupervised morphology induction using word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Henning Sperr, Jan Niehues, and Alex Waibel. 2013. Letter n-gram-based input encoding for continuous space language models. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1).

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th Inter-national Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May.

John Wieting, Mohit Bansal, Kevin Gimpel, Karen Livescu, and Dan Roth. 2015. From paraphrase database to compositional paraphrase model and back. *Transactions of the ACL (TACL)*.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Towards universal paraphrastic sentence embeddings. In *Proceedings of International Conference on Learning Representations*.

Wei Xu, Chris Callison-Burch, and William B Dolan. 2015. SemEval-2015 task 1: Paraphrase and semantic similarity in Twitter (PIT). In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval)*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*.

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. 2015. Long short-term memory over recursive structures. In *Proceedings of the 32nd International Conference on Machine Learning*.